Chương 2:
Quản lý bộ nhớ

Biến: tên biến, vùng nhớ và giá trị biến

- Mỗi biến trong C có tên và giá trị tương ứng. Khi một biến được khai báo, một vùng nhớ trong máy tính sẽ được cấp phát để lưu giá trị của biến. Kích thước vùng nhớ phu thuộc kiểu của biến, ví dụ 4 byte cho kiểu int

```
int x = 10;
```

- Khi lệnh này được thực hiện, trình biên dịch sẽ thiết lập để 4 byte vùng nhớ này lưu giá trị 10.
- Phép toán & trả về địa chỉ của *x*, nghĩa là địa chỉ của ô nhớ đầu tiên trong vùng nhớ lưu trữ giá trị của *x*.

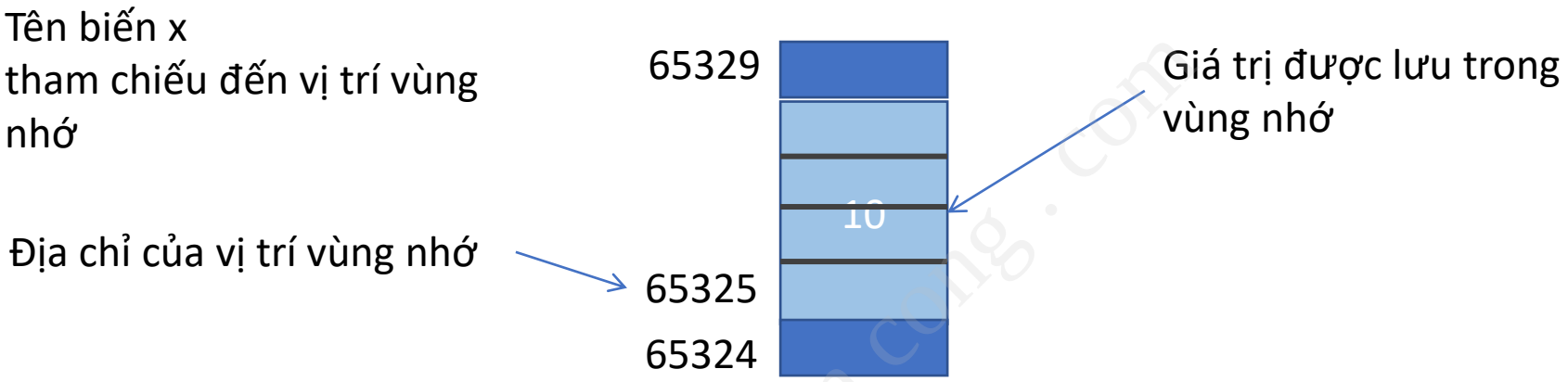
Biến: tên biến, vùng nhớ và giá trị biến

- Mỗi biến trong C có tên và giá trị tương ứng. Khi một biến được khai báo, một vùng nhớ trong máy tính sẽ được cấp phát để lưu giá trị của biến. Kích thước vùng nhớ phu thuộc kiểu của biến, ví dụ 4 byte cho kiểu int

```
int x = 10;
```

- Khi lệnh này được thực hiện, trình biên dịch sẽ thiết lập để 4 byte vùng nhớ này lưu giá trị 10.
- Phép toán & trả về địa chỉ của *x*, nghĩa là địa chỉ của ô nhớ đầu tiên trong vùng nhớ lưu trữ giá trị của *x*.

Biến: tên biến, vùng nhớ và giá trị biến



- `int x = 10;`
- `x` biểu diễn tên biến (e.g. an assigned name for a memory location)
- `&x` biểu diễn địa chỉ ô nhớ đầu tiền thuộc vùng nhớ của `x`, tức là 65325
- `*(&x)` or x biểu diễn giá trị được lưu trong vùng nhớ của `x`, tức là 10

Chương trình ví dụ

```
#include<stdio.h>
int main() {
   int x = 10;
   printf("Value of x is %d\n",x);
   printf("Address of x in Hex is %p\n",&x);
   printf("Address of x in decimal is
  %lu\n",&x);
   printf("Value at address of x is
  %d\n",*(&x));
   return 0;
}
```

```
Value of x is 10
Address of x in Hex is 0061FF0C
Address of x in decimal is 6422284
Value at address of x is 10
```

Khái niệm con trỏ

- Con trỏ là một biến chứa địa chỉ vùng nhớ của một biến khác.

- Cú pháp chung khai báo biến con trỏ

```
data_type* ptr_name;
```

- Ký hiệu '*' thông báo cho trình biên dịch rằng *ptr_name* là một biến con trỏ và *data_type* chỉ định rằng con trỏ này sẽ trỏ tới địa chỉ vùng nhớ của một biến kiểu *data_type*.

Tham chiếu ngược (dereference) biến con trỏ

- Chúng ta có thể “tham chiếu ngược" một con trỏ, nghĩa là lấy giá trị của biến mà con trỏ đang trỏ vào bằng cách dùng pháp toán ‘`*`’, chẳng hạn `*ptr`.

- Do đó, `*ptr` *có giá trị 10*, vì 10 đang là giá trị của *x*.

Ví dụ con trỏ

```
int x = 10;

int *p = &x;
```

Nếu con trỏ `p` giữ địa chỉ của biến `a`, ta nói `p` trỏ tới biến `a`

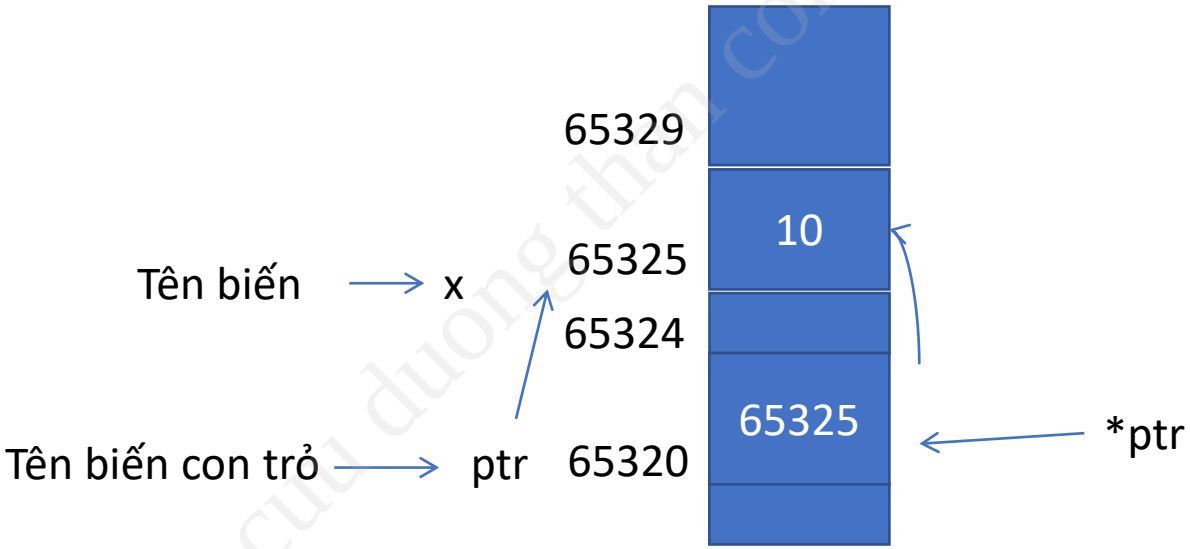
`*p` biểu diễn giá trị lưu tại địa chỉ `p`

`*p` được gọi là "tham chiếu ngược" của con trỏ `p`

- Con trỏ được dùng để lấy địa chỉ của biến nó trỏ vào, đồng thời cũng có thể lấy giá trị của biến đó

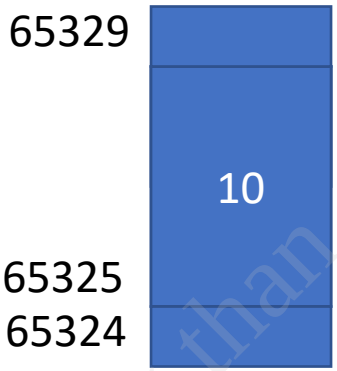
Con trỏ

```
int x = 10;
int *ptr;
ptr = &x;
```



Biến và con trỏ

Biến
int x = 10;
&x biểu diễn địa chỉ của x
x biểu diễn giá trị lưu tại địa chỉ &x



Con trỏ
int *p; p được khai báo là con trỏ int
p = &a; p giữ giá trị địa chỉ a, i.e.65325
*p biểu diễn giá trị lưu tại p, ie.10
int b = *p; b có giá trị 10.
*p = 20; a có giá trị 20.

```
int *p;
*p = 10; // this is not correct.
```

Ví dụ con trỏ

```
#include<stdio.h>
int main() {
  int x = 10;
  int *ptr;
  ptr = &x;
  printf("Value of x is %d\n", x);
  printf("Address of x is %lu\n", (unsigned long int) &x);
  printf("Value of pointer ptr is %lu\n", (unsigned long int)
  ptr);
  printf("Address of pointer ptr is %lu\n", (unsigned long int)
  &ptr);
  printf("Ptr pointing value is %d\n",*ptr);
  return 0;
}
```

```
Value of x is 10
Address of x is 6422284
Value of pointer ptr is 6422284
Address of pointer ptr is 6422280
Ptr pointing value is 10
```

Các phép toán trên con trỏ

- Cộng hoặc trừ với 1 số nguyên n trả về 1 con trỏ cùng kiểu, là địa chỉ mới trỏ tới 1 đối tượng khác nằm cách đối tượng đang bị trỏ n phần tử
- Trừ 2 con trỏ cho ta khoảng cách (số phần tử) giữa 2 con trỏ
- KHÔNG có phép cộng, nhân, chia 2 con trỏ
- Có thể dùng các phép gán, so sánh các con trỏ
  - Chú ý đến sự tương thích về kiểu.

Ví dụ

```
char *pchar; short *pshort; long *plong;
pchar ++; pshort ++; plong ++;
```

Giả sử các địa chỉ ban đầu tương ứng của 3 con trỏ là 100, 200 và 300, kết quả ta có các giá trị 101, 202 và 304 tương ứng

Nếu viết tiếp

```
plong += 5; => plong = 324
pchar -=10; => pchar = 91
pshort +=5; => pshort = 212
```

Chú ý

`++` và `--` có độ ưu tiên cao hơn `*` nên `*p++` tương đương với `*(p++)` tức là tăng địa chỉ mà nó trỏ tới chứ không phải tăng giá trị mà nó chứa.

`*p++ = *q++` sẽ tương đương với

```
*p = *q;
p=p+1;
q=q+1;
```

Con trỏ *void

- Là con trỏ không định kiểu. Nó có thể trỏ tới bất kì một loại biến nào.
- Thực chất một con trỏ `void` chỉ chứa một địa chỉ bộ nhớ mà không biết rằng tại địa chỉ đó có đối tượng kiểu dữ liệu gì. Do đó không thể truy cập nội dung của một đối tượng thông qua con trỏ `void`.
- Để truy cập được đối tượng thì trước hết phải ép kiểu biến trỏ void thành biến trỏ có định kiểu của kiểu đối tượng

Con trỏ *void

```
float x; int y;
void *p; // khai báo con trỏ void
p = &x; // p chứa địa chỉ số thực x
*p = 2.5; // báo lỗi vì p là con trỏ void
/* cần phải ép kiểu con trỏ void trước khi
truy cập đối tượng qua con trỏ */
*((float*)p) = 2.5; // x = 2.5
p = &y; // p chứa địa chỉ số nguyên y
*((int*)p) = 2; // y = 2
```

Con trỏ và mảng

- Giả sử ta có `int a[30];` thì `&a[0]` là địa chỉ phần tử đầu tiên của mảng đó, đồng thời là địa chỉ của mảng.
- Trong C, tên của mảng chính là một hằng địa chỉ bằng địa chỉ của phần tử đầu tiên của mảng

```
a = &a[0];
a+i = &a[i];
```

Con trỏ và mảng

Tuy vậy cần chú ý rằng `a` là 1 hằng nên không thể dùng nó trong câu lệnh gán hay toán tử tăng, giảm như `a++;`

- Xét con trỏ: `int *pa;`

```
pa = &a[0];
```

Khi đó pa trỏ vào phần tử thứ nhất của mảng và

`pa +1` sẽ trỏ vào phần tử thứ 2 của mảng

`*(pa+i)` sẽ là nội dung của `a[i]`

Con trỏ và xâu

- Ta có `char tinhthanh[30] =“Da Lat”;`
- Tương đương :

```
char *tinhthanh;
tinhthanh=“Da lat”;
```

Hoặc: `char *tinhthanh =“Da lat”;`

- Ngoài ra các thao tác trên xâu cũng tương tự như trên mảng

```
*(tinhthanh+3) = “l”
```

- Chú ý : với xâu thường thì không thể gán trực tiếp như dòng thứ 3

Mảng các con trỏ

- Con trỏ cũng là một loại dữ liệu nên ta có thể tạo một mảng các phần tử là con trỏ theo dạng thức.

```
<kiểu> *<mảng con trỏ>[số phần tử];
```

Ví dụ: `char *ds[10];`

  - `ds` là 1 mảng gồm 10 phần tử, mỗi phần tử là 1 con trỏ kiểu char, được dùng để lưu trữ được của 10 xâu ký tự nào đó

- Cũng có thẻ khởi tạo trực tiếp các giá trị khi khai báo

```
char * ma[10] = {"mot","hai","ba"...};
```

Mảng các con trỏ

- Một ưu điểm khác của mảng trỏ là ta có thể hoán chuyển các đối tượng (mảng con, cấu trúc..) được trỏ bởi con trỏ này bằng cách hoán đổi các con trỏ
- Ưu điểm tiếp theo là việc truyền tham số trong hàm
- Ví dụ: Vào danh sách lớp theo họ và tên, sau đó sắp xếp để in ra theo thứ tự ABC.

```
#include <stdio.h>
#include <string.h>
#define MAXHS  50
#define MAXLEN 30
```

Mảng các con trỏ

```
int main () {
  int  i, j, count = 0;   char ds[MAXHS][MAXLEN];
  char *ptr[MAXHS],  *tmp;
  while ( count < MAXHS) {
      printf(" Vao hoc sinh thu : %d  ",count+1);
      gets(ds[count]);
      if (strlen(ds[count] == 0) break;
      ptr[count] = ds +count;
      ++count;
   }
   for ( i=0;i<count-1;i++)
      for ( j =i+1;j < count; j++)
         if (strcmp(ptr[i],ptr[j])>0) {
           tmp=ptr[i]; ptr[i] = ptr[j]; ptr[j] = tmp;
   }
   for (i=0;i<count; i++)
      printf("\n %d :  %s", i+1,ptr[i]);
}
```

Con trỏ trỏ tới con trỏ

- Bản thân con trỏ cũng là một biến, vì vậy nó cũng có địa chỉ và có thể dùng một con trỏ khác để trỏ tới địa chỉ đó.

```
<Kiểu dữ liệu> ** <Tên biến trỏ>;
```

- Ví dụ:

```
int x = 12;
int *p1 = &x;
int **p2 = &p1;
```

- Có thể mô tả một mảng 2 chiều qua con trỏ của con trỏ theo công thức :

```
M[i][k] = *(*(M+i)+k)
```

Trong đó:

- `M+i` là địa chỉ của phần tử thứ `i` của mảng
- `*(M+i)` cho nội dung phần tử trên
- `*(M+i)+k` là địa chỉ phần tử `[i][k]`

Con trỏ trỏ tới con trỏ

Ví dụ: in ra một ma trận vuông và cộng mỗi phần tử của ma trận với 10

```
#include <stdio.h>
#define hang 3
#define cot 3
int main() {
  int mt[hang][cot] = {{7,8,9},
                       {10,13,15},
                       {2,7,8}};
  int i,j;
  for (i=o; i<hang; i++) {
     for (j=0; j<cot; j++)
       printf(" %d ", mt[i][j]);
     printf("\n");
   }
```

```
for (i=0; i<hang; i++) {
     for (j=0; j<cot; j++) {
       *(*(mt+i)+j)=*(*(mt+i)+j) +10;
       printf(" %d ", *(*(mt+i)+j);
     }
     printf("\n"); }
 }
```

Quản lý bộ nhớ - Bộ nhớ động

- Cho đến lúc này ta chỉ dùng bộ nhớ tĩnh: tức là khai báo mảng, biến và các đối tượng khác một cách tường minh trước khi thực hiện chương trình.
- Trong thực tế nhiều khi ta không thể xác định trước được kích thước bộ nhớ cần thiết để làm việc, và phải trả giá bằng việc khai báo dự trữ quá lớn
- Nhiều đối tượng có kích thước thay đổi linh hoạt

Quản lý bộ nhớ - Bộ nhớ động

- Việc dùng bộ nhớ động cho phép xác định bộ nhớ cần thiết trong quá trình thực hiện của chương trình, đồng thời giải phóng chúng khi không còn cần đến để dùng bộ nhớ cho việc khác
- Trong C ta dùng các hàm `malloc, calloc, realloc` và `free` để xin cấp phát, tái cấp phát và giải phóng bộ nhớ.
- Trong C++ ta dùng `new` và `delete`

Cấp phát bộ nhớ động

- Cú pháp xin cấp phát bộ nhớ:

```
<biến trỏ> = new <kiểu dữ liệu>;
```

hoặc

```
<biến trỏ> = new <kiểu dữ liệu>[số phần tử];
```

dòng trên xin cấp phát một vùng nhớ cho một biến đơn, còn dòng dưới cho một mảng các phần tử có cùng kiểu với kiểu dữ liệu.

- Giải phóng bộ nhớ

```
delete ptr; // xóa 1 biến đơn
delete [] ptr; // xóa 1 biến mảng
```

Cấp phát bộ nhớ động

- Bộ nhớ động được quản lý bởi hệ điều hành, được chia sẻ giữa hàng loạt các ứng dụng, vì vậy có thể không đủ bộ nhớ. Khi đó toán tử new sẽ trả về con trỏ NULL
- Ví dụ:

```
int *ptr; ptr = new int [200];
if (ptr == NULL) {
    // thông báo lỗi và xử lý
}
```

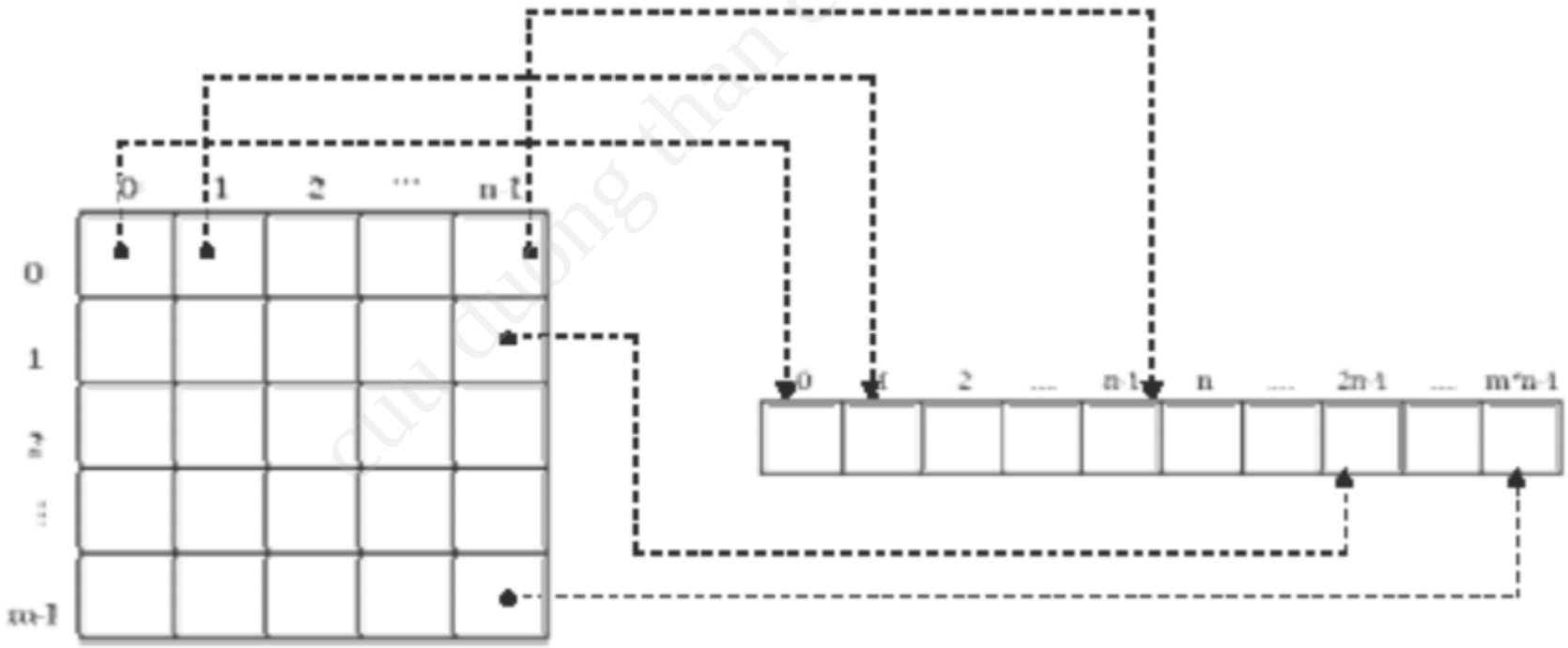
Ví dụ

```
#include <stdio.h>
int main() {
    int i,n; long total=100,x,*ds;
    printf(" Vao so ptu "); scanf("%d",&n);
    ds = new long [n];
    if (ds==NULL) exit(1);
    for (i=0;i<n;i++){
        printf("\n Vao so thu  %d : ", i+1 );
        scanf("%d",&ds[i] );
    }
    printf("Danh sach cac so : \n");
    for (i=0;i<n;i++)   printf("%d",ds[i]);
        delete []ds;
        return 0;
    }
```

Bộ nhớ động cho mảng 2 chiều

- Cách 1: Biểu diễn mảng 2 chiều thành mảng 1 chiều
- Gọi X là mảng hai chiều có kích thước m dòng và n cột. A là mảng một chiều tương ứng, khi đó

```
X[i][j] = A[i*n+j]
```



Bộ nhớ động cho mảng 2 chiều

- Cách 2: Dùng con trỏ của con trỏ
- Ví dụ: Với mảng số nguyên 2 chiều có kích thước là R * C ta khai báo như sau:

```
int **mt;
mt = new int *[R];
int *temp = new int[R*C];
for (i=0; i< R; ++i) {
   mt[i] = temp;
   temp += C;
}
```

- Để giải phóng:

```
delete [] mt[0];
delete [] mt;
```

Bộ nhớ động cho mảng 2 chiều

- Ví dụ khác để cấp phát động cho mảng hai chiều chứa các số thực `float`

```
// Khởi tạo ma trận với
// R hàng và C cột
 float  ** M = new float *[R];
 for (i=0; i<R;i++)
    M[i] = new float[C];
// Dùng M[i][j] cho
// các phần tử của ma trận
```

```
// Giải phóng
 for(i=0; i<R;i++)
    // Giải phóng các hàng
    delete []M[i];
 delete []M;
```